Understanding and Mitigating Refresh Overheads in High-Density DDR4 DRAM Systems

Janani Mukundan
Computer Systems Laboratory
Cornell University
Ithaca, NY USA
mukundan@csl.cornell.edu

Hillery Hunter
IBM Thomas J. Watson
Research Center
Yorktown Heights, NY USA
hhunter@us.ibm.com

Kyu-hyoun Kim
IBM Thomas J. Watson
Research Center
Yorktown Heights, NY USA
kimk@us.ibm.com

Jeffrey Stuecheli
IBM Systems and Tech. Group
Austin, TX USA
jeffas@us.ibm.com

José F. Martínez
Computer Systems Laboratory
Cornell University
Ithaca, NY USA
martinez@cornell.edu

ABSTRACT

Recent DRAM specifications exhibit increasing refresh latencies. A refresh command blocks a full rank, decreasing available parallelism in the memory subsystem significantly, thus decreasing performance. Fine Granularity Refresh (FGR) is a feature recently announced as part of JEDEC's DDR4 DRAM specification that attempts to tackle this problem by creating a range of refresh options that provide a trade-off between refresh latency and frequency.

In this paper, we first conduct an analysis of DDR4 DRAM's FGR feature, and show that there is no one-size-fits-all option across a variety of applications. We then present *Adaptive Refresh (AR)*, a simple yet effective mechanism that dynamically chooses the best FGR mode for each application and phase within the application.

When looking at the refresh problem more closely, we identify in high-density DRAM systems a phenomenon that we call *command queue seizure*, whereby the memory controller's command queue seizes up temporarily because it is full with commands to a rank that is being refreshed. To attack this problem, we propose two complementary mechanisms called *Delayed Command Expansion (DCE)* and *Preemptive Command Drain (PCD)*.

Our results show that AR does exploit DDR4's FGR effectively. However, once our proposed DCE and PCD mechanisms are added, DDR4's FGR becomes redundant in most cases, except in a few highly memory-sensitive applications, where the use of AR does provide some additional benefit. In all, our simulations show that the proposed mechanisms yield 8% (14%) mean speedup with respect to traditional refresh, at normal (extended) DRAM operating temperatures, for a set of diverse parallel applications.

1. INTRODUCTION

The dynamic nature of DRAM requires logic to track carefully the DRAM lines that need to be refreshed, and to issue refresh commands in a timely fashion. In the past, refresh commands were relatively short and infrequent, and thus system performance and power were not significantly impacted. As DRAM density increased and more bits needed to be refreshed, the industry decided to design for a constant per-cell retention time (64 ms) and refresh interval (t_{REFI}), changing instead the time that each refresh command takes to complete (t_{RFC}). Essentially, a strategic decision was made to refresh a larger number of rows per refresh command, rather than issuing more refresh commands.

As t_{RFC} increases, concern has been raised about the performance impact of refresh [8, 17]. When a refresh command is issued, it blocks a full rank, decreasing available parallelism in the memory subsystem significantly. Memory requests to the rank currently being refreshed stall until the refresh command completes, affecting system performance. To attack this problem, recent approaches rely on issuing refresh commands less frequently than specified by the DRAM manufacturer [8], or scheduling refresh commands at opportune times [17]. (We comment on related work in Section 3.)

1.1 DDR4 DRAM and Fine Granularity Refresh

First discussed in US Patent 2012/0151131 [7], a Fine Granularity Refresh (FGR) feature has been recently announced as part of JEDEC's DDR4 DRAM specification [1]. FGR attempts to tackle increases in t_{RFC} by creating a range of refresh options for memory controller use: 1x refresh is a direct extension of DDR2 and DDR3 refresh: each refresh command takes t_{RFC} ns, and it must be issued every t_{REFI}=7.8 μs. 2x and 4x modes require that refresh commands be issued twice and four times as frequently–at 3.9 and 1.95μs intervals, respectively. However, in these modes, fewer DRAM rows are refreshed per command, and as a result, their refresh latencies $t_{\text{RFC\_2x}}$ and $t_{\text{RFC\_4x}}$ are shorter (although not proportionally). Table 1 lists the FGR parameters specified for DDR4 DRAM.

1.2 Contributions

This paper makes the following contributions:

– We conduct an analysis of DDR4 DRAM's FGR feature, and determine that there is no single mode that works well across all the applications studied.

– We propose *Adaptive Refresh*, a simple yet highly effective mech-

ISCA'13, June 23–27, 2013, Tel-Aviv, Israel.

Table 1: Refresh cycle times (t_{RFC} = amount of time each refresh command takes) and refresh intervals (t_{REFI} = how frequently refresh commands must be issued) in the DDR4 DRAM specification [1]. Values for large chips are extrapolated.

Chip Size	t_{RFC_1x}	t_{RFC_2x}	t_{RFC_4x}
8 Gb	350 ns	260 ns	160 ns
16 Gb	480 ns	350 ns	260 ns
32 Gb	640 ns	480 ns	350 ns
t_{REFI}	7.8 μs	3.9 μs	1.95 μs

anism that leverages FGR, by dynamically choosing the mode that best suits each application, and each phase within the application.

– We introduce *command queue seizure*, a phenomenon that we show will become a concern in systems built with high-density DRAM chips (16 and 32 Gb). Command queue seizure occurs when the memory controller issues a refresh command to a rank, only to find that it is soon unable to process memory requests to other ranks because the command queue is taken up by commands associated with the rank being refreshed (and are thus ineligible for processing).

– We propose *Delayed Command Expansion* (DCE) and *Preemptive Command Drain* (PCD), two complementary mechanisms to address the command queue seizure phenomenon. In DCE, the memory controller withholds expansion of memory requests into the command queue if such requests are to ranks that are currently being refreshed. In PCD, the scheduler prioritizes commands in the command queue that map to ranks about to be refreshed, so that they may leave the queue and not occupy valuable slots during the upcoming refresh cycle. The net result is a higher availability of the command queue to memory requests that can proceed concurrently to specific refresh actions.

Our results show that AR does exploit DDR4's FGR effectively. However, once our proposed DCE and PCD mechanisms are added, FGR becomes redundant in most cases, except in a few highly memory-sensitive applications, where the use of AR does provide some additional benefit. In all, AR+DCE+PCD yields 8 and 14% performance gains with respect to traditional refresh at normal and extended DRAM operating temperatures, respectively.

2. BACKGROUND

Modern DRAM-based memory systems are organized with many degrees of parallelism. Each microprocessor chip has one or more memory controllers, which control one or more *channels*. Each channel has a small number of *ranks*, where each rank contains a handful of DRAM chips (including ECC). Within each chip are a handful of *banks*. All chips in a DRAM rank respond in tandem to commands, meaning a memory controller has scheduling parallelism opportunities across banks, ranks, and channels.

For server memory systems, total system capacity is a primary design objective. Logically speaking, a memory controller may be able to support as many as 32 ranks (2 LRDIMMs with 2 physical ranks/DIMM and 8-high DRAM stacks per rank). However, electrically, channels are much more constrained, due to I/O channel signaling limitations. The faster a channel is run, the more power is burned in signaling and termination. In this paper we simulate an energy-efficient server-class configuration: a four-rank system at a power-friendly data rate of 1,600 Mbps. In the future, lower-power, higher-frequency offerings may be possible through 3D TSV-Stacked Master-Slave (3DS) technology [6].

2.1 JEDEC DDR4 DRAM Specification

The initial JEDEC DDR4 DRAM specification was released in September 2012 [1], and initial server products with DDR4 DRAM chips are anticipated to begin shipping in 2014 [20]. There are several key new features of this standard: memory speeds are expanded to 3,200 Mbps (compared to 1,600 Mbps for the initial DDR3 specification) with high-tap (V_{DDQ}) termination; core and I/O supply voltage (V_{DD}/V_{DDQ}) are lowered to 1.2 V (compared to 1.5 V for DDR3 and 1.35 V for DDR3L) with a new supply (V_{PP}) added for word-line voltage; several reliability features are added, such as write data CRC, command/address bus parity, and dynamic bus inversion; and architecturally, a key change is the concept of a *bank group*, with the number of banks increased from eight (DDR3) to sixteen. Two (x16) to four (x4/x8) bank groups per chip allow interleaving of column access operations between different bank groups at the periphery blocks (data lines and data sense amplifier) keeping column cycle time (t_{CCD}) to the minimum timing (4 cycles). To help with core power, the page size for x4 parts will be cut in half, dropping from 1 kByte to 512 Bytes. Several standby power reduction features are also added, such as gear down mode, chip-select-to-address latency, and maximum power saving mode.

2.2 DDR4 DRAM Refresh Challenges

As technology scaling advances, DRAM tends to maintain the same number of electrons in its storage capacitor (Yoon and Tressler [21] estimate on the order of 100,000), so unlike NAND technology, which will soon reach tens of electrons per floating-gate cell [21], DRAM can theoretically continue to scale. However, concerns about the end of scaling remain well founded, since it will become increasingly difficult to maintain the same amount of charge in the storage capacitor in future technologies. One major reason is that the per-cell capacitance will decrease as lithographic scaling results in physically smaller DRAM cells, but the aspect ratio of the deep-trench cell capacitor cannot keep growing. Supply voltage scaling to meet power reduction demands also causes charge loss. Increased series resistance of the cell access transistor and the bit line (due to smaller cell geometry) makes restoration of charge into the cell slower, and leaves insufficient charge in the storage capacitor. In addition to a decrease in charge, the input offset voltage of the sense amplifier also increases due to increased variability in its transistor pair, which makes it more difficult to sense data, even if the amount of charge is the same.

All of these scaling issues make it more difficult to meet JEDEC's DRAM cell retention time specification of 64 ms in future technology. Thus, in addition to increases in t_{RFC} as DRAM chip density increases, it is anticipated that t_{REFI} may also worsen.

To tackle the refresh problem, DDR4 includes several new concepts, one of which we leverage in this paper: Fine Granularity Refresh. In addition, DDR4 includes low power auto self-refresh and temperature-controlled refresh mode, but these are related to saving refresh power at low temperature and when idling, and are not within the scope of this work.

3. RELATED WORK

RAIDR – Liu et al. propose RAIDR [8], a clever attempt to minimize refresh operations by exploiting inter-cell variation in retention time. The authors assert that only a small number of weak cells require a conservative refresh interval of 64 ms. DRAM rows are grouped into several retention bins, based on the measured minimum retention time across all cells in the corresponding row. Rows are then refreshed at different rates based on the bin they are classified in.

While this may be well suited to certain domains, we believe there are four important reasons why caution is in order when operating DRAMs outside the standards specified mechanisms for server systems, where data integrity is frequently a non-negotiable design constraint.

– Corner sensitivity: The manifestation of a retention failure is dependent not just on the amount of charge retained in a DRAM cell capacitor, but also on the ability of the sensing mechanism to distinguish a 0 from a 1. Retention characteristics are therefore highly dependent on the combination of temperature, voltage, and DRAM internal noise encountered during a particular test, because the effects which must be considered are not just at the DRAM cell, but also in the sensing mechanism. While chip temperature is something which can be measured during an in-system chip characterization run, a system operator generally has little control over the voltages and noise in the memory subsystem, and no ability to know the internal noise sensitivities of a part (since these will be highly design-dependent). The weak-cell conclusions of a particular characterization pass may thus not represent worst-case retention characteristics, and it is very challenging for a system-level test to control voltage supply variability and internal DRAM noise.

– Manufacturer freedom: JEDEC specifications only dictate a command called "Refresh" along with its duration, necessary frequency, and how to measure the current consumed during this operation (I_{dd5}). Manufacturers are actually free to do whatever needs to be done to maintain the DRAM during the t_{RFC} duration of the refresh command. From this perspective, an assumption that a refresh command may only refresh some rows, or may only refresh a particular number of rows during t_{RFC}, is technically unsafe. While these assumptions may be accurate for certain DRAMs, they are not guaranteed to apply over all DRAMs across manufacturers, and across the lifetime of any standard.

– Access-pattern-dependent mechanisms: A DRAM cell's or row's retention time of its contents may not only be related to the last time at which it was written, but may also be sensitive to the data pattern which is stored or the access patterns to surrounding cells.

– VRT: Variable Retention Time is not a single phenomenon, but rather a general characterization of DRAMs, which states that there are factors which cause some percentage of DRAM cells to exhibit different retention characteristics at different points in time (e.g., exposure to very high temperature during component assembly). VRT effects occur both during manufacturing and in the field. While ECC and other error-tolerance mechanisms are designed to handle intermittent errors, extending the interval at which cells are refreshed introduces the risk that total error accumulation, due to VRT, standard retention, soft errors, and other effects, may rise above the initial design threshold for a system.

The combination of these four factors means that it may be risky to operate DRAM cells at a refresh interval beyond the specification range that DRAM manufacturers guarantee using their own test methodology. We believe it is crucial to address the refresh challenge through alternate means, for those systems where data integrity is essential. For systems where the risk of weak-cell migration or incorrect characterization is acceptable, the work presented in this paper is largely complementary to prior refresh interval extension approaches, and can be used in conjunction with those techniques.

Elastic Refresh – JEDEC specifications allow some flexibility in issuing refresh commands: up to eight refresh commands can be postponed or issued in advance. Stuecheli et al. [17] propose Elastic Refresh, a technique that effectively exploits the flexible dynamic range allowed in the JEDEC refresh specification, by being less aggressive in issuing refresh commands. The primary idea behind elastic refresh is to use predictive mechanisms that decrease the probability of a read or a write operation from colliding with a refresh operation. Earlier techniques make use of the flexibility in issuing refresh operations by scheduling them any time the bus or the rank queues are idle. The elastic refresh algorithm extends this concept by waiting an additional period after the rank becomes idle before it issues the refresh operation. This additional idle delay not only reduces the priority of the refresh command, but also exploits bursty behavior in applications.

Although Elastic Refresh lowers the priority of refresh commands and tries to reduce collisions with reads and writes, as DRAM scales, the increase in t_{RFC} effectively reduces the probability of avoiding such collisions. In this sense, our Adaptive Refresh mechanism is better suited for adapting to bursty behavior in systems and avoiding collisions by moving between DDR4 DRAM's 1x and 4x modes. Additionally, the concept of Elastic Refresh can be easily integrated with our proposed Adaptive Refresh scheme.

Smart Refresh and Selective DRAM Refresh – Ghosh et al. [5] and Song [16] propose similar techniques that eliminate unnecessary DRAM refresh commands and overheads. The Smart Refresh algorithm proposed in [5] maintains a "refresh-needed" counter for each row that gets reset every time the row gets read out or written to. Smart Refresh relies heavily on data access patterns of the workloads, as the number of refreshes issued only reduces if a large number of rows are being activated. Additionally, it has a high area overhead: for a 2 GB DRAM module, it requires 131,072 counters, each 3-bit wide. Song's Selective DRAM Refresh [16] proposes the use of a reference bit for each DRAM row, which is set when being accessed. During refresh, if a row's reference bit is set, refresh is skipped. This proposal suffers from similar issues as Smart Refresh.

RAPID and Flikker – RAPID, proposed by Venkatesan et al. [18], is a software solution that exploits retention time variation among different DRAM cells. The primary idea is to allocate pages with longer retention time before those with shorter retention time. This allows selection of a refresh period that is dependent on the shortest allocated page retention. RAPID risks similar problems as RAIDR, caused by dynamic variation in retention time and DRAM scaling. In addition, its performance is based on the utilization of the memory pages. Flikker [9], another software solution, proposes partitioning data into critical and non-critical groups. Non-critical data is then refreshed at much lower rates than critical data. However, identifying data criticality requires substantial programmer effort and is orthogonal to our Adaptive Refresh mechanism.

4. UNDERSTANDING DDR4 DRAM'S FINE GRANULARITY REFRESH (FGR)

The internal operation of DRAM during refresh includes activating some number of pages, waiting for data to be fully restored, and then precharging those pages. The number of pages that are refreshed together depends on the device density, and can be calculated as $(\text{Density}/\text{PageSize}) \times (t_{\text{REFI}}/64\ \text{ms})$. This corresponds to 256 pages on a 16 Gb x8 DDR4 DRAM chip for the 1x mode.

Activating a large number of pages simultaneously generates more sensing current than the internal regulator or the power delivery network can sustain. Therefore, refresh pages are internally divided into subgroups, and refresh is performed per subgroup sequentially, with some time delay between subgroups to reduce the

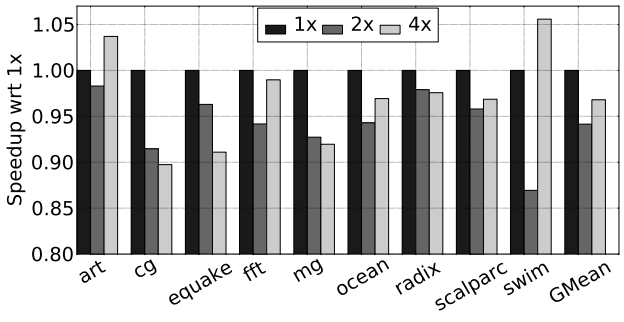

Figure 1: Performance (higher is better) of the 1x, 2x and 4x modes, running in the normal temperature range (below 85°C), normalized to the 1x mode.

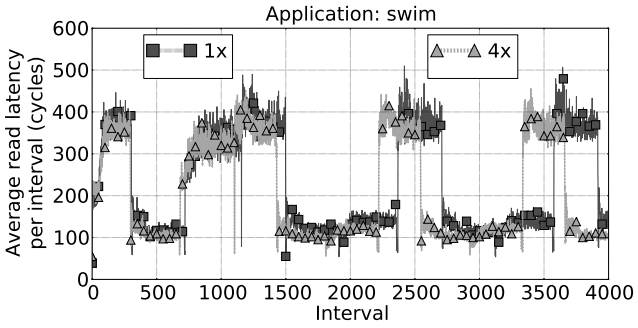

Figure 2: Average read latency for *swim* in 1x and 4x FGR modes (lower is better for performance).

peak noise current. For example, 256 pages can be divided into 32 subgroups, with each group having 8 pages, and the 32 subrefresh operations are staggered every 10 ns (manufacturer and technology dependent).

For a normal activation-precharge operation, the minimum row cycle time is denoted as t_{RC} cycles, and determines the minimum time between accesses to different rows; but for refresh operations, the cycle time is longer than t_{RC}, as each sub-refresh operation activates more than one page and generates more sensing noise. We denote this as $t_{RC\_Refresh}$. When the subrefresh staggering delay t_{STAG} is added, the time to complete one refresh operation in 1x mode can be expressed as $t_{RFC\_1x} = (N-1) \times t_{STAG} + t_{RC\_Refresh}$, where N represents the number of subgroups.

For the 2x or 4x modes, there are N/2 or N/4 subgroups, respectively. Thus, the time to complete one refresh operations in 2x mode is $t_{RFC\_2x} = ((N/2)-1) \times t_{STAG} + t_{RC\_Refresh}$, and similarly for 4x mode.

Note that t_{RFC_2x} is longer than half of t_{RFC_1x}, and that t_{RFC_4x} is also longer than half of t_{RFC_2x}, both due to the term $t_{RC\_Refresh}$. In other words, for each subrefresh operation, there is a startup and completion overhead time $t_{RC\_Refresh}$ which must be amortized. For the 2x and 4x modes, this is amortized over a smaller number of refreshes, so the total time spent doing refreshes in 2x and 4x mode grows, as this initiation cost is paid 2x and 4x more frequently. This introduces a tradeoff: total DRAM stall time due to refresh is minimized when long-latency (many-row) 1x mode is used, but the average stall time should be smallest when shorter refresh operations are used, allowing arriving reads to be issued as soon as possible.

4.1 FGR Characterization

To understand the impact of FGR on performance, we run experiments on a set of parallel applications using the 1x, 2x and 4x refresh modes. (A detailed description of the experimental methodology can be found in Section 7.) Figure 1 shows the corresponding performance data. From the plot we see that, on average, the 1x mode tends to perform better than both the 2x and 4x modes. However, for certain applications like *art* and *swim*, the 4x mode tends to perform better than both 1x and 2x modes. We also observe that 2x is always either the second best or the worst performing among all modes. The reason is because, when moving from 1x to 2x to 4x, while t_{REFI} scales linearly, t_{RFC} doesn't, and this makes the 2x mode sub-optimal. Therefore, for the remaining part of the paper we do not consider the 2x mode for our experiments

In order to gain insight as to why certain applications performed better in the 1x mode and others performed better in the 4x mode, we choose two applications to analyze: *swim* (which has better 4x performance) and *equake* (which has better 1x performance). We find that for memory-intensive applications, there is a clear benefit from short-latency refresh operations (4x mode). This is because the command queue will often receive requests for a rank that is being refreshed, and these commands will sit in the queue waiting for refresh to complete. The result is a longer average DRAM access time. Figure 2 shows the average read latency for *swim* (from the SPEC-OMP parallel suite) when running in 1x and 4x modes. 4x mode has on average a smaller read latency than the 1x mode, because DRAM commands spend less time waiting for a refresh to complete in the 4x mode, therefore improving performance.

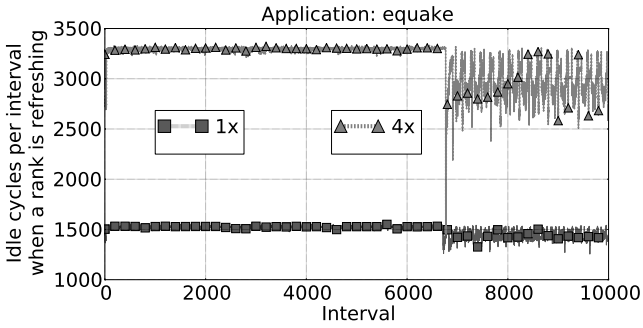

Figure 3: Number of cycles the controller remains idle while a rank is refreshing (lower is better for performance) for *equake* in 1x and 4x FGR modes.

On the other hand, for applications with low memory utilization, longer refresh commands are less disruptive to overall performance. The cumulative time spent doing refreshes is lower for long-latency commands–more rows are refreshed per command, thus better amortizing $t_{RC\_Refresh}$. Figure 3 shows the average number of cycles the memory controller remains idle due to refresh for *equake* (also from the SPEC-OMP parallel suite) when running the 1x and 4x refresh mode configurations. The 4x mode yields more idle cycles than the 1x mode, which affects performance negatively.

5. ADAPTIVE REFRESH

From our above analysis, we conclude that no single refresh mode works best in all cases. Thus, we set out to design an adaptive FGR mechanism capable of determining the best refresh mode for a particular workload during the course of its execution.

We propose a very simple Adaptive Refresh mechanism to achieve this goal. The mechanism tracks data bus utilization as a proxy

measurement for performance, because it is directly observable at the memory controller, and because it tends to correlate strongly with system performance for memory-bound applications. Specifically, at each interval, we use a counter that is incremented every time the memory controller issues a read or a write—the two commands that involve an actual data transfer.

Algorithm 1 Simple FGR-Aware Adaptive Refresh

```
1: while application is still running do
2:     Choose 1x mode and run the application for n intervals, while monitoring data bus utilization
3:     Choose 4x mode and run the application for n intervals, while monitoring data bus utilization
4:     if utilization in 1x mode ≥ utilization in 4x mode then
5:         Choose 1x mode and run the application for m ≫ n intervals
6:     else
7:         Choose 4x mode and run the application for m ≫ n intervals
8:     end if
9: end while
```

Algorithm 1 shows the simple procedure for our Adaptive Refresh mechanism. We start by initially running the application in the 1x mode for a training period of n intervals, while monitoring data bus bandwidth. At the end of n intervals, we switch from the 1x mode to the 4x mode, and train for a period of another n intervals, again while monitoring data bus bandwidth. At the end of these $2n$ intervals, we compare the measured utilization for both refresh modes, and pick the mode that has yielded the higher utilization. We then continue to run using the chosen mode for a period of $m \gg n$ intervals. This whole process is repeated periodically in order to accommodate changes in phase behavior of the application.

Picking the right n, m, and interval duration is important. In our experiments, we empirically determined the refresh rate for the 1x mode (t_{REFI}) to be a good value for our algorithm's interval. This is convenient because a refresh command is sent out every t_{REFI} cycles in the 1x mode (twice/four times as often for the 2x/4x modes), so picking this interval trivially guarantees that the controller does not miss any refreshes, even when switching modes. As for n and m, we empirically determine that $n = 5$ and $m = 100$ is a good compromise when all the associated overheads are accounted for. We evaluate this Adaptive Refresh mechanism in Section 8.1.

Micro-architectural Support for Adaptive Refresh

Very few minor additions are required to a memory scheduler in order to incorporate Adaptive Refresh. Note that a scheduler already has registers that store the current values of t_{RFC} and t_{REFI}. Two additional registers are required to store the value of the training and testing intervals: n (3-bit register) and m (7-bit register). Another 7-bit register and a 7-bit adder are needed to keep track of the elapsed intervals during the training and testing phases. Modern-day memory controllers already have the capability of measuring data bus utilization, and therefore no additional hardware is required to monitor the bandwidth of the data bus. However, a 13-bit adder and two 15-bit registers are required to keep track of the cumulative utilization across n intervals during the two training phases. Finally, a 15-bit comparator is necessary to compare the utilization measured during training in the 1x and 4x modes and a 2-to-1 multiplexor is needed to choose the FGR mode based on the output of the comparator.

6. INCREASING THE COMMAND QUEUE EFFECTIVENESS OF HIGH - DENSITY DRAM

During a typical refresh operation, the controller cannot issue any DRAM commands to a rank while it is being refreshed. Fortunately, modern server memory systems have multiple ranks, and thus when a refresh operation to a rank is underway, the controller can still issue DRAM commands to the other ranks that are not being refreshed. In earlier technology generations like DDR3 DRAM, the refresh latency t_{RFC} was still small enough that issuing DRAM commands to non-refreshing ranks could likely keep the controller busy. However, for very dense DRAM chips, this may no longer be the case.

Recall that, as DRAM's chip density increases, the time required to refresh these DRAM chips also increases. As refresh latency begins to grow, we notice that our modeled controller remains idle for a longer period of time despite the presence of multiple ranks in the memory system. In this section, we introduce and analyze a new phenomenon caused by refresh commands on the memory controller: *command queue seizure*.

For a memory system designed with multiple ranks, refreshes are staggered across the ranks. This is done primarily for two reasons: (1) Issuing simultaneous refreshes to all ranks can push memory systems close to (or over) their peak power budgets (or cause voltage drop issues). This is because refresh is the most power-hungry DRAM operation, and the V_{dd} voltage rail is commonly shared across all chips in the system. Although not specifically reported in DDR JEDEC specifications, almost all the modern multi-rank memory controllers of which we are aware stagger refreshes among ranks for this reason (because it is a system design issue involving power budgets). (2) From a performance perspective, staggering refreshes to multiple ranks ensures that the controller does not remain idle while a rank is being refreshed. Due to the complexity involved in the design of command queues for on-chip memory controllers in current memory systems, a majority of system designs opt for a common command queue for all requests targeting any DRAM rank on a given memory channel [14].[1] For a high-performance memory system, addresses will be hashed across channels and ranks, enabling maximum system parallelism. An understanding of the impact of refresh on the command queue can be obtained by looking at Figures 4 and 5. These plots show the effect of command queue seizure when running a micro-benchmark with an even distribution of commands arriving to each rank, for a system modeled using 4 Gb and 32 Gb DDR4 DRAM chips.

In Figure 4, we see the command queue occupancy and the number of reads being issued per cycle for a two-rank and four-rank system using a 4 Gb chip. Let us first analyze the two-rank system. In interval 3120, a refresh is issued to rank 0. It remains idle for the duration of refresh while the scheduler issues commands to the other rank in the system, rank 1. However, due to the even distribution of the memory stream, the controller will steadily drain the command queue of rank 1 requests, while filling it with requests to rank 0. When the command queue gets filled up with requests to rank 0, the controller stalls, as it can no longer issue any more commands. This is indicated by the plateaus in the "# reads" line, which tracks the cumulative count of issued read operations to all locations in the memory system. For a four-rank system using 4 Gb chips, this poses less of a problem, as the extra ranks add sufficient variety of commands to the queue's available scheduling resources.

[1] Additionally, experiments conducted using per-rank command queues (as opposed to per-channel command queues) did not provide a significant improvement in performance to warrant the need.

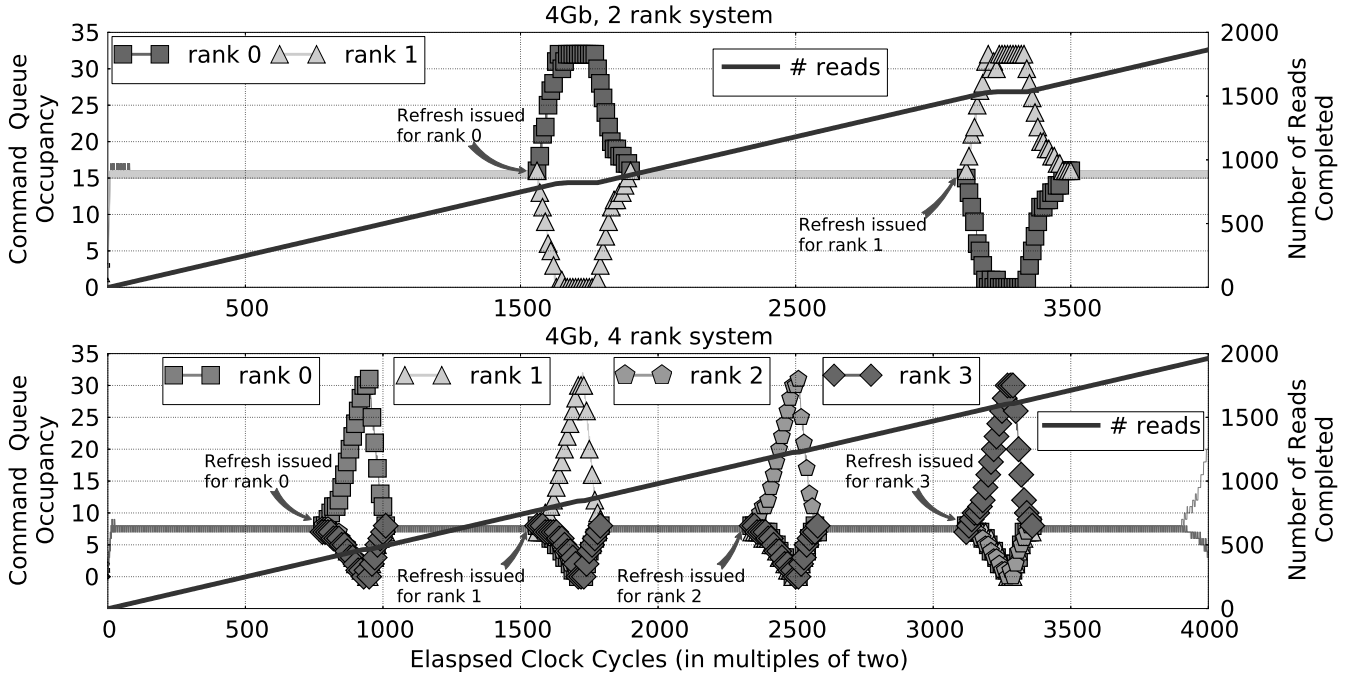

Figure 4: Analysis of the command queue seizure phenomenon for a 4 Gb DRAM chip running a micro-benchmark with an even distribution of loads and stores across ranks and banks. For a two-rank system, the command queue can fill up with DRAM commands to a rank being refreshed, momentarily stalling command issue and increasing the idle time of the scheduler, thereby hurting performance. For a four-rank system, the problem is alleviated with sufficient command variety in the queue, and the controller is able to continue to issue commands while a refresh operation completes in a target rank.

The queue seldom fills up with pending commands to the rank being refreshed, and the controller is continuously able to issue operations to the memory system ("# reads" increases monotonically).

The situation changes significantly if we consider the increased refresh cycle times of high density DRAMs (32 Gb chips) (figure 5). Here the increased refresh latency (t_{RFC}) blocks the rank being refreshed for a longer period of time. This gives the command queue more time to fill up with commands to the rank being refreshed. The situation becomes untenable for high-density memory with longer refresh latencies, and causes the controller to stall for relatively long periods of time. The phenomenon actually worsens for a four-rank system when compared to a two-rank system, as the number of refresh commands sent in one refresh interval (t_{REFI}) is also twice as many.

6.1 Preemptive Command Drain

In this section we propose a new scheduling technique that helps mitigate the negative effects of command queue seizure, which we call *Preemptive Command Drain (PCD)*. The idea behind PCD is to drain the command queue of commands to a rank that is about to be refreshed, by prioritizing them over commands to other ranks. This is easily accomplished by checking whether the refresh countdown of each DRAM rank is below a certain threshold. In this way, when the refresh operation is actually issued by the controller, there will be fewer (possibly none) commands to the refreshing rank in the command queue, and more commands to non-refreshing ranks. This gives the memory controller more opportunities for successful command scheduling while the refresh operation is taking place, thereby reducing idle cycles.

For this to be effective, it is important to pick the right refresh countdown threshold. In the general case, we could use a simple interval adaptation scheme, similar to the one proposed for Adaptive Refresh (Section 5), to determine the right threshold for each application and each application phase. In our experimental setup, however, we conducted a sensitivity study, and found that a threshold of 150-250 cycles was universally optimal across the studied applications and phases. Thus, without losing generality, in our evaluation we show PCD results with a constant threshold of 200 cycles (Section 8.2).

Micro-architectural Support for PCD

Most modern memory schedulers use some variant of the FR-FCFS scheduling algorithm proposed by Rixner et al. [12]. The basic FR-FCFS scheduling policy prioritizes CAS commands over RAS commands and to break ties, older over younger commands. In order to incorporate PCD into FR-FCFS, a few minor changes to the scheduling algorithm would be needed. PCD + FR-FCFS would need to prioritizes commands in the following order: (a) CAS commands over RAS commands for the soon-to-be-refreshed rank; (b) older commands over younger commands for the soon-to-be-refreshed rank; (c) CAS commands over RAS commands for other ranks; and finally (d) older commands over younger commands for other ranks. An 8-bit comparator would be required to check if the refresh countdown of a DRAM rank is below a particular threshold. If so, commands mapping to this rank would then be selected with higher priority.

6.2 Delayed Command Expansion

Memory controllers typically hold loads and stores received from the last-level cache in a transaction queue. As space becomes available in the command queue, these memory requests are expanded into the appropriate DRAM commands and transferred to the com-

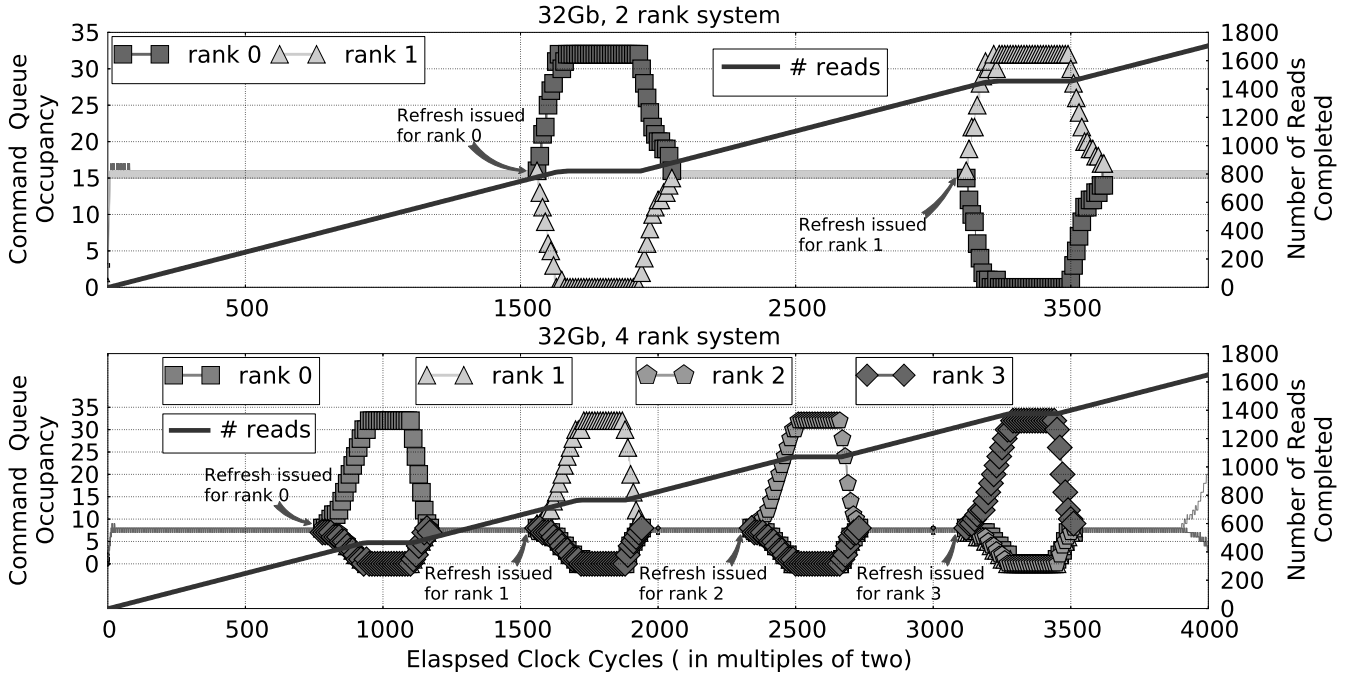

Figure 5: Analysis of the command queue seizure phenomenon for a 32 Gb DRAM chip running a micro-benchmark with an even distribution of loads and stores across ranks and banks. In large capacity DRAM chips, for a two-rank system, once the command queue has been filled with commands to the rank being refreshed, long $t_{\mathbf{RFC}}$ times quickly leads to the memory scheduler stalling for longer. The problem worsens for four-rank systems as the number of refresh commands issued per refresh interval doubles. The net result is an increase in the idle time of the scheduler, which leads to a loss in performance.

mand queue. The PCD algorithm explained above tackles the negative effects of refresh on the command queue, by prioritizing the scheduling of commands to a rank about to be refreshed, so they can be drained from the command queue instead of getting stuck in place for the duration of the refresh operation. This process can also be optimized on the transaction queue side: We propose that the memory controller be allowed to temporarily put off memory requests to a rank that is being refreshed, instead expanding and transferring to the command queue requests to non-refreshing ranks. This helps increase the number of issuable commands inside the command queue, potentially improving performance. We call this Delayed Command Expansion (DCE), and evaluate it in Section 8.2.

Micro-architectural Support for DCE

In our model, when memory requests arrive at the transaction queue, they are expanded into DRAM commands in FIFO order and placed in the command queue. In order to incorporate DCE, an additional comparison of the rank id's of the memory requests with the current rank that is being refreshed would also be needed. As a conservative estimate, this requires a 2-bit comparator for each transaction queue entry. However, it is also possible for multiple entries to share a single comparator, as long as they are used serially. DCE is triggered only when a rank is refreshing. Any memory controller would already have this information stored, and hence additional hardware would not be required to support this. Similar to PCD, the information of whether a rank is being refreshed or not would be determined using a comparator, and if so commands mapping to this rank would then be selected with lower priority for expansion into the command queue.

7. EXPERIMENTAL METHODOLOGY

7.1 Architecture Model

Our baseline processor model integrates eight cores and supports a DDR4-1600 memory subsystem with one independent memory channel (we model our memory subsystem based on the configurations used in IBM Power7™ systems 710, 720 and 730, where the ratio of threads to memory controllers is eight [14]). The DIMM structure and timing parameters of our memory model follow JEDEC's DDR4 SDRAM specification [1]. For power and energy calculations, we use I_{dd} values estimated from discussions with authors of the JEDEC DDR4 standard and DRAM chip manufacturers. The micro-architectural features of the baseline processor are shown in Table 2; the parameters of the L2 cache, the memory system, and the DRAM power model are shown in Tables 3 and 4.

7.2 Simulation Setup and Applications

All our experiments have been carried out by extending the SESC simulation environment [11] with DRAMSim2 [13], which has been modified to model JEDEC's DDR4 specification, including bank and rank groups, and Fine Granularity Refresh. We evaluate a number of configurations as follows: The configuration 1x represents the baseline auto-refresh policy present in current day memory systems; the 4x configuration makes use of the corresponding FGR refresh mode; AR is our proposed adaptive refresh technique that leverages FGR; PCD denotes our proposed Preemptive Command Drain mechanism, wherein the scheduler prioritizes commands in the command queue that are mapped to ranks that are about to be refreshed; and finally, DCE is our proposed Delayed Command Expansion scheme, that withholds expansion of memory requests into the command queue if they are to a refreshing rank. We evalu-

ate our proposed schemes on a set of parallel applications, running eight threads each, to completion. Our parallel workloads constitute a good mix of scalable scientific programs from different benchmark suites, as shown in Table 5.

7.3 DDR4 Extended Temperature Range

In the normal operating temperature range for DRAMs (below 85°C), the average time between refresh commands (t_{REFI}) is 7.8 μs. DRAMs can also operate in the extended temperature range (between 85°C and 95°C), particularly in server type environments and while using 3DS technology [6, 15]. The required time between refresh commands at high temperatures is halved to 3.9 μs. There is global interest in expanding the operating temperature ranges of data centers, driven mostly by the desire for achieving higher operating efficiency and lower total cost of ownership [2]. Increasing operating temperature ranges in a data center environment causes all components in servers (including memory) to see higher temperatures. Evaluation of the extended temperature range is hence both necessary and critical when evaluating high-density memory systems.

Table 2: Core Parameters.

Technology	32 nm
Frequency	3.2 GHz
Number of cores	8
Fetch/issue/commit width	4/4/4
Int/FP/Ld/St/Br Units	2/2/2/2/2
Int/FP Multipliers	1/1
Int/FP issue queue size	32/32 entries
ROB (reorder buffer) entries	96
Int/FP registers	96 / 96
Ld/St queue entries	24/24
Max. unresolved br.	24
Br. mispred. penalty	9 cycles min.
Br. predictor	Alpha 21264 (tournament)
RAS entries	32
BTB size	512 entries, Direct-mapped
iL1/dL1 size	32 kB
iL1/dL1 block size	32 B/32 B
iL1/dL1 round-trip latency	2/3 cycles (uncontended)
iL1/dL1 ports	1 / 2
iL1/dL1 MSHR entries	16/16
iL1/dL1 associativity	Direct-mapped/4-way
Memory Disambiguation	Perfect
Coherence protocol	MESI
Consistency model	Release consistency

8. EVALUATION

8.1 Adaptive Refresh

Figure 6 shows the performance obtained by the 1x, 4x, and AR configurations in the normal DRAM operating temperature range, normalized to the performance of the 1x configuration. Our proposed AR essentially matches the performance of the refresh mode that works best for each individual application. For applications that are not as memory-sensitive (which tend to work better in the 1x configuration), like *mg*, *cg*, *equake* and *radix*, AR outperforms the 4x configuration significantly and is within less than 2% of the 1x configuration. For more memory-sensitive applications like *art*, AR performs better than the 1x configuration, and it is within 1.5% of the performance of the 4x mode. For some applications like *fft*, *ocean*, *scalparc*, and *swim*, AR outperforms both the 1x and the 4x configurations, as AR successfully adapts to application phase changes.

Analysis – Let's look into what is happening. We first take an example from the class of applications that are less memory sen-

Table 3: Parameters of the shared L2 and DRAM.

Shared L2 Cache Subsystem	
Shared L2 Cache	4 MB, 64 B block, 8-way
L2 MSHR entries	64
L2 round-trip latency	32 cycles (uncontended)
Write buffer	64 entries
DDR4 @1600 Mbps DRAM – 16Gb chip size	
Transaction Queue	128 entries
Command Queue	32 entries
Number of Channels	1
DIMM Configuration	Quad rank
Number of Banks	16 per rank
Row Buffer Size	1 KB
Address Mapping	Page Interleaving
Row Policy	Closed Page[a]
Burst Length	8
t_{RCD}	10 DRAM cycles
t_{CL}	10 DRAM cycles
t_{WL}	12 DRAM cycles
t_{CCD}	4 DRAM cycles
t_{CCD_L}	5 DRAM cycles
t_{WTR}	2 DRAM cycles
t_{WTR_L}	6 DRAM cycles
t_{WR}	15 DRAM cycles
t_{RTP}	6 DRAM cycles
t_{RP}	10 DRAM cycles
t_{RRD}	4 DRAM cycles
t_{RTRS}	2 DRAM cycles
t_{RAS}	28 DRAM cycles
t_{RC}	28 DRAM cycles
t_{FAW}	20 DRAM cycles
t_{CKE}	4 DRAM cycles
Refresh Parameters: DDR4 @1600 Mbps DRAM – 16Gb chip size	
t_{REFI}	7.8 μs
$t_{REFI\text{-}XTemp}$	3.9 μs
t_{RFC_1x}	384 DRAM cycles
t_{RFC_2x}	280 DRAM cycles
t_{RFC_4x}	208 DRAM cycles

[a]We have conducted our experiments with open page policy and the results and insights closely follow those for closed page, which we present.

Table 4: Parameters used for 16 Gb DDR4 @ 1600 Mbps DRAM power management features.

IDD0	24 mA
IDD1	32 mA
IDD3P	7.2 mA
IDD2P	6.4 mA
IDD2N	10.1 mA
IDD3N	16.6 mA
IDD4R	60 mA
IDD4W	58 mA
IDD5	102 mA
IDD6	6.7 mA
IDD7	107 mA

Table 5: Simulated parallel applications and their input sets.

Data Mining		[10]
scalparc	Decision Tree	125k pts., 32 attributes
NAS OpenMP		[4]
mg	Multigrid Solver	Class A
cg	Conjugate Gradient	Class A
SPEC OpenMP		[3]
swim-omp	Shallow water model	MinneSPEC-Large
equake-omp	Earthquake model	MinneSPEC-Large
art-omp	Self-organizing Map	MinneSPEC-Large
Splash-2		[19]
ocean	Ocean movements	514×514 ocean
fft	Fast Fourier transform	1M points
radix	Integer radix sort	2M integers

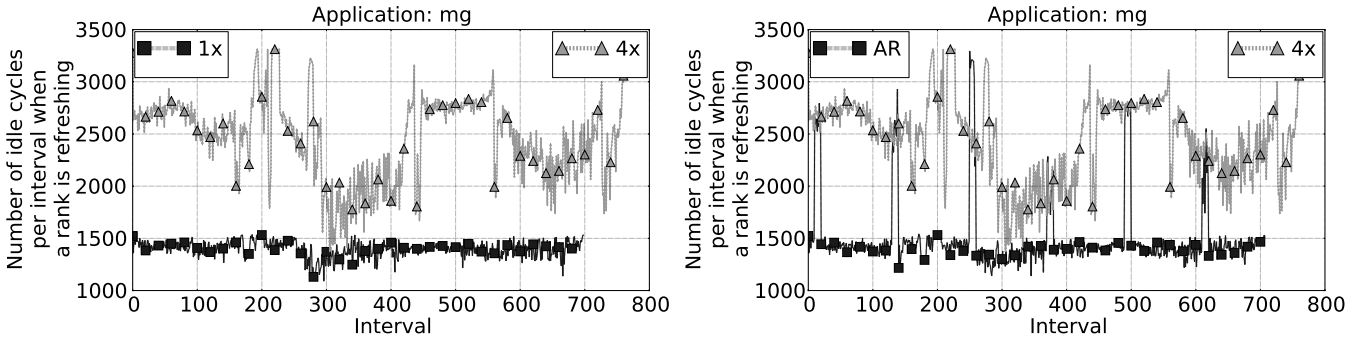

Figure 7: Number of cycles per interval the controller remains idle (lower is better) while a rank is refreshing and the command queue is not empty for the application *mg*. The plot to the left shows idle cycles for the 1x and 4x FGR configurations. The plot to the right shows the same for the 4x mode and our proposed AR configuration. AR closely follows the FGR mode that has the fewest scheduler idle cycles (1x in this case), which translates to improved performance.

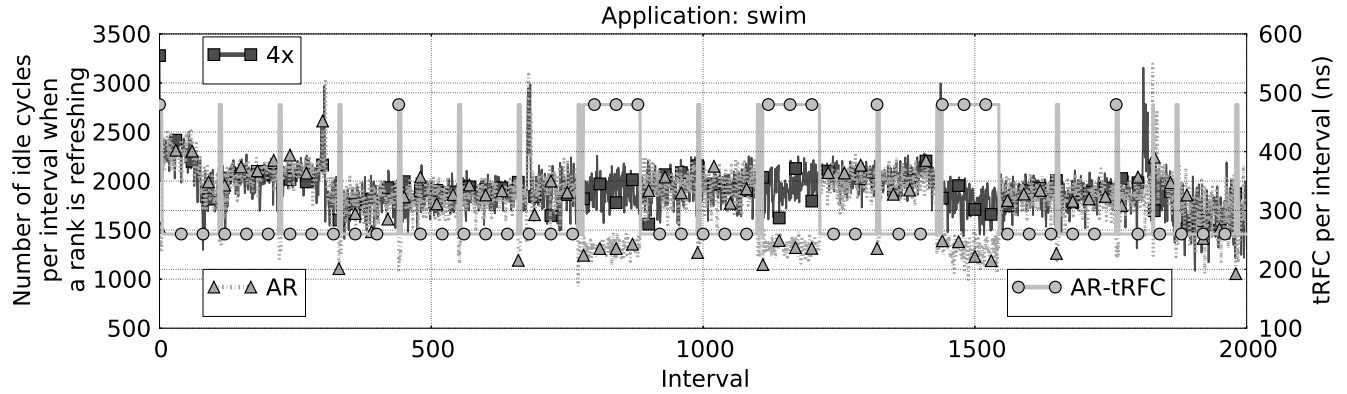

Figure 8: Number of cycles per interval the controller remains idle (lower is better) while a rank is refreshing and the command queue is not empty for the application *swim* when running the 4x and AR configurations. The plot shows that AR closely tracks the 4x mode for the most part, but also has periodic drops in idle cycles. These drops correspond to a profitable switch from the 4x to the 1x FGR mode due to a phase change in the application, as indicated by the overlaid t_{RFC} plot which jumps to 480ns from 260ns (when the switch occurs).

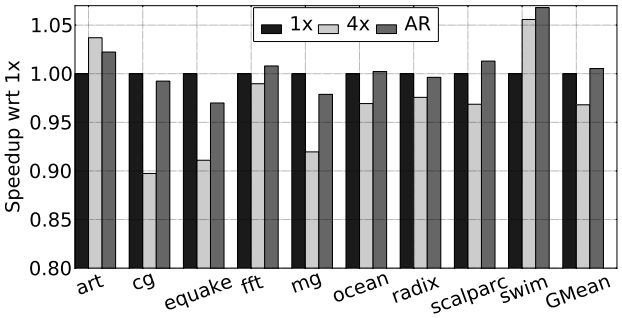

Figure 6: Performance (higher is better) in the normal DRAM temperature range for the 1x, 4x and AR configurations, normalized to the performance of the 1x configuration.

sitive: *mg*. Figure 7 is divided into two plots: The left plot shows the number of cycles per interval the memory controller remains idle while a rank is refreshing (lower is better), in spite of the command queue not being empty, when running the 1x and 4x configurations. The right plot shows the same for AR and the 4x configurations. From the plot on the left, we see that the 1x configuration has fewer scheduler idle cycles than the 4x configuration, which translates into improved system performance. We can see that AR (right plot) closely tracks the behavior of the 1x configuration. The right plot does show some periodic, narrow spikes for the AR configuration. These correspond to AR's training phase in 4x mode. As it turns out, *mg* doesn't have phase changes that alter the refresh mode patterns significantly, and 4x mode is never picked by AR. Fortunately, since the training phase is a small fraction of the overall execution, it affects system performance minimally, as shown earlier.

Next we look at an example from the class of applications for which AR performs better than both 1x and 4x modes: *swim*. As before, Figure 8 shows the number of cycles per interval the memory controller remains idle while a rank is refreshing (lower is better), when the command queue is not empty. Similar to *mg*, the plot shows that AR closely tracks the better configuration, 4x in this case. However this time AR exhibits wide, periodic drops in idle time. These correspond to adaptation to program phase changes, where AR concludes that a temporary switch to 1x mode is profitable. The overlaid plot of t_{RFC} shows that this is indeed what is happening. The net result is better performance than the static 4x configuration, as shown before (Figure 6).

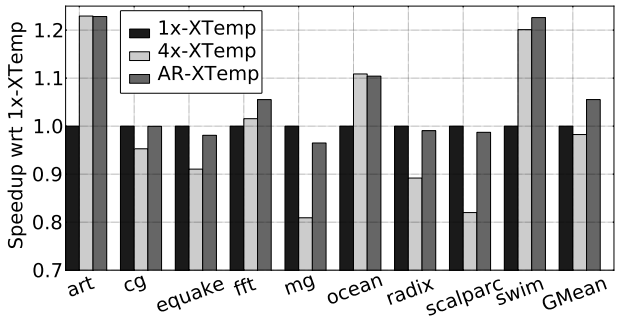

Figure 9: Performance (higher is better) in the extended DRAM temperature range for the 1x, 4x and AR configurations, normalized to the performance of the 1x configuration.

Extended DRAM temperature range – Figure 9 shows the performance obtained by the 1x, 4x and AR configurations in extended DRAM operating temperature range, normalized to the performance of the 1x configuration. (Recall that operating in extended DRAM temperature range is an important consideration for high-performance configurations.) In the extended temperature range (XTemp), the volume of refresh commands issued essentially doubles within the same refresh interval. This means that the controller spends more time performing refresh operations. The results show the same trends as in the earlier case of normal DRAM temperature mode, only the performance benefit of picking the right refresh mode is larger, and thus AR is even more beneficial.

8.2 Delayed Command Expansion and Preemptive Command Drain

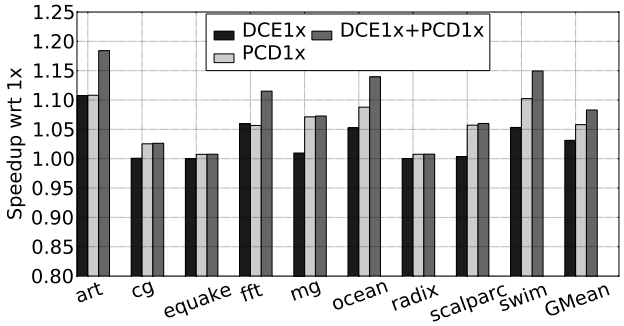

Figure 10: Performance (higher is better) in the normal DRAM temperature range for DCE, PCD and DCE+PCD when running in the 1x mode, normalized to the performance of the 1x configuration.

Figure 10 shows the performance obtained by the 1x, Delayed Command Expansion in 1x mode (DCE1x), Preemptive Command Drain in 1x mode (PCD1x), and DCE1x+PCD1x configurations, run under normal DRAM operating temperature conditions, and normalized to the performance of the 1x configuration. We see that DCE1x and PCD1x improve performance on average by 3% and 5.5% over the 1x configuration, respectively. Moreover, combining the two schemes has an additive effect: DCE1x+PCD1x shows an improvement in performance of 8% over the 1x configuration.

Analysis – To understand where this improvement is coming from, let's look at *art* a bit more closely. Recall that both the DCE and PCD mechanisms attempt to increase the number of commands to non-refreshing ranks in the command queue, by proactively draining the queue of commands to the refreshing rank, and by prioritizing commands to non-refreshing ranks. Figure 11 shows the percentage of command queue slots taken up by commands to non-refreshing ranks per interval for the application *art* when running the 1x and DCE1x+PCD1x configurations. The results show that the configuration DCE1x+PCD1x has a much higher number of commands to non-refreshing ranks in the command queue per interval, when compared to the 1x configuration. This increases the opportunity for issuing more commands per interval, reducing idle cycles in the controller and improving throughput, ultimately improving performance.

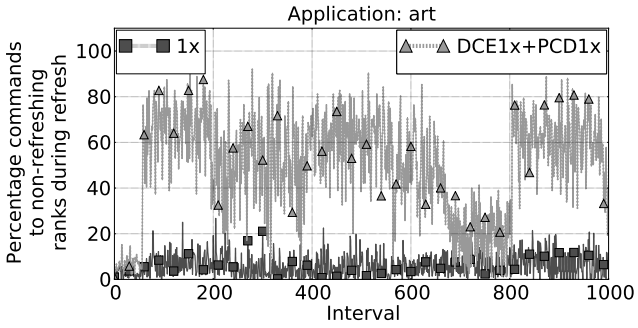

Figure 11: Fraction of commands to non-refreshing ranks in the command queue (higher is better) while a rank is being refreshed for the application *art* when running the 1x configuration and the DCE+PCD configuration in the 1x mode.

8.3 Putting It All Together: AR+DCE+PCD

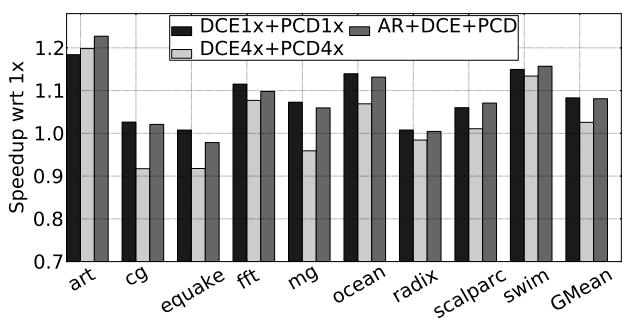

Figure 12: Performance (higher is better) in the normal DRAM temperature range when running DCE+PCD in the 1x, 4x and AR modes, normalized to the performance of the 1x configuration.

Figures 12 and 13 show the performance obtained by combining our three proposed schemes when running in both normal and extended DRAM temperature ranges, and normalized to the appropriate 1x mode in each case. A few interesting observations can be made from these plots: First, for most applications, adding DCE and PCD to the 4x mode reduces performance when compared to adding the schemes to the 1x mode. This is because the 4x mode cumulatively spends more time on refresh than the 1x mode. We notice that the increase in refresh commands while running the 4x mode increases the idle time during refresh, especially for those applications that do not provide a constant stream of loads and stores (which helps DCE and PCD). Second, following from the above

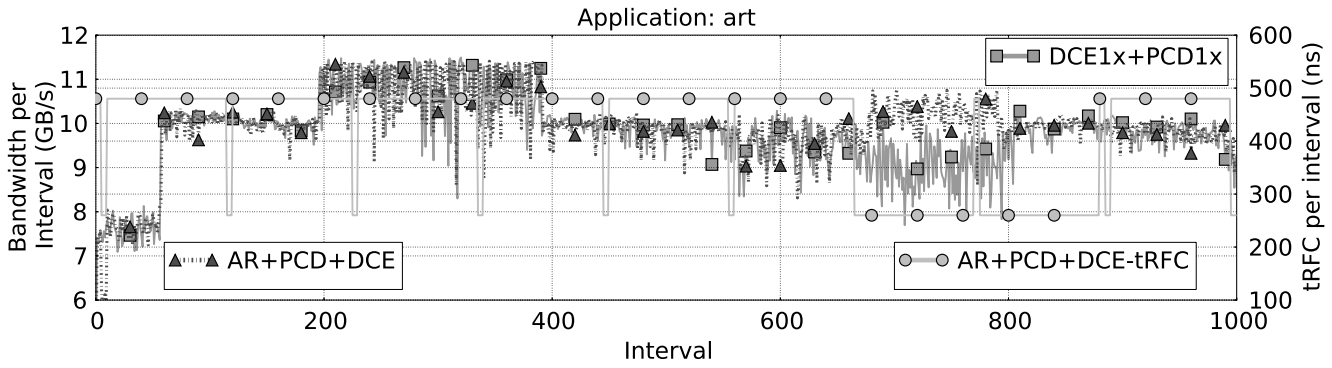

Figure 14: Effective data bandwidth for the application *art* when running DCE+PCD in the 1x and AR configurations. AR closely follows the 1x configuration for the most part, but also makes profitable switches to the 4x mode when it finds an opportunity for increasing data bus utilization. This is indicated by the the overlaid t_{RFC} plot in the AR configuration which jumps to 260ns from 480ns when these switches occur.

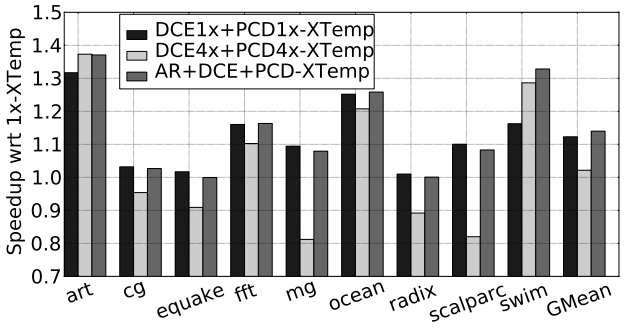

Figure 13: Performance (higher is better) in the extended DRAM temperature range when running DCE+PCD in the 1x, 4x and AR modes, normalized to the performance of the 1x configuration.

argument, adding AR to DCE and PCD provides no significant improvement in performance when compared to adding the schemes to the 1x mode. The combination of AR, DCE, and PCD, improves performance over the 1x configuration by 8 and 14% on average in normal and extended DRAM temperature ranges, respectively. These results are remarkably close (in fact, virtually identical in the case of normal DRAM temperature range) to the ones obtained by DCE1x+PCD1x alone. What this means is that, on average for these applications, leveraging DDR4 DRAM's Fine Granularity Refresh feature offers little advantage once our proposed DCE and PCD mechanisms are in place. For applications that are particularly memory sensitive and/or exhibit refresh phase behavior (*art* and *swim*), we do observe an improvement by adding AR on top of DCE and PCD, especially in the extended DRAM temperature range.

Analysis – To provide further insight, we look at the access pattern of the application *art*, one of the two that benefit from combining AR to DCE+PCD. Figure 14 shows the effective data bandwidth per interval for the DCE1x+PCD1x and the AR+DCE+PCD configurations on the primary y-axis (left), and t_{RFC} per interval for the AR+DCE+PCD configuration on the secondary y-axis (right). (For DCE1x+PCD1x, t_{RFC} remains constant at 480 ns.) Around the 650 interval time frame, the application *art* has a memory phase change. AR+DCE+PCD detects this change and immediately switches to the 4x mode. DCE1x + PCD1x, however, remains in the 1x mode. As a result, AR + DCE + PCD is able to sustain a high effective data bandwidth, whereas DCE1x+PCD1x drops in bandwidth at that point.

8.4 Energy Calculations

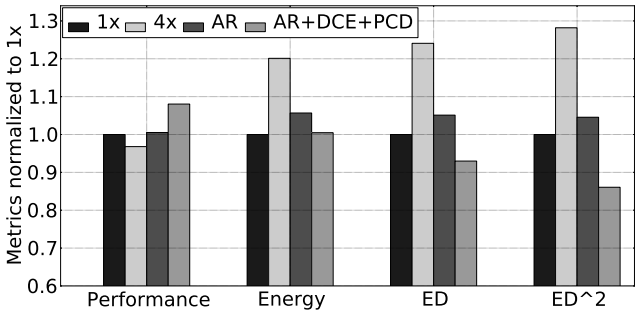

Figure 15: Mean performance, energy, energy-delay and energy-delay squared in the normal DRAM temperature range for the 1x, 4x, AR and AR+DCE+PCD configurations, normalized to that of the 1x configuration.

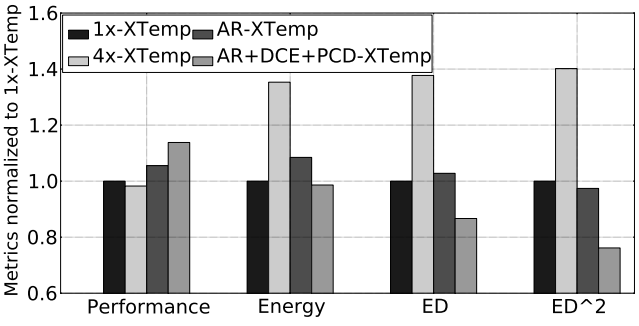

Figure 16: Mean performance, energy, energy-delay and energy-delay squared in the extended DRAM temperature range for the 1x, 4x, AR and AR+DCE+PCD configurations, normalized to that of the 1x configuration.

Figures 15 and 16 show average performance, energy, energy-delay and energy-delay squared numbers for the 1x, 4x, AR, and

AR+DCE+PCD configurations, normalized to that of the 1x configuration in each case, in both normal and extended DRAM temperature ranges. In the normal DRAM temperature range, AR+DCE+ PCD improves performance by 8% on average, and reduces energy-delay and energy-delay squared by 7 and 14%, respectively. In the extended DRAM temperature range, AR+DCE+ PCD improves performance by 14% on average, and reduces energy-delay and energy-delay squared by 14 and 24%, respectively. As expected, because of the increase in the volume of refresh operations, the 4x configuration exhibits the highest energy consumption. Thus, our proposed AR+DCE+PCD not only improves performance significantly, it does so by consuming the same amount of energy when compared to the 1x configuration, for an overall improvement in energy efficiency.

8.5 Comparison to RAIDR

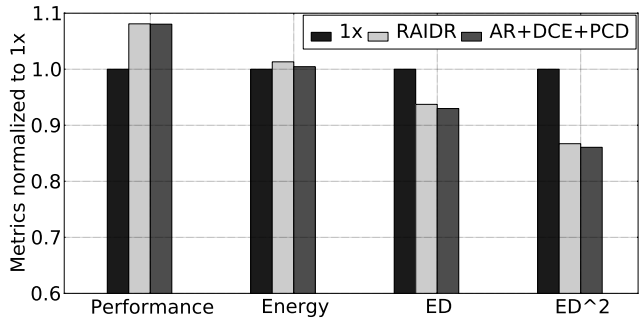

Figure 17: Average performance, energy, energy-delay, and energy-delay squared in the normal DRAM temperature range for the 1x, RAIDR and AR+DCE+PCD configurations, normalized to that of the 1x configuration.

Figure 17 shows the performance, energy, energy delay, and energy delay squared numbers for the the 1x mode, RAIDR [8], and AR+DCE+PCD. RAIDR is a RAS-only refresh implementation that cuts down significantly on the number of refreshes by exploiting inter-cell variation in retention, resulting in improved performance and energy efficiency (Section 3). The plot shows that AR +DCE+PCD virtually matches RAIDR in every performance and energy metric. This is significant because it does so (1) without resorting to RAIDR's RAS-only refresh (which bypasses DRAM's auto-refresh feature and requires the controller to identify on the address bus which bank needs to be refreshed at each point in time), and (2) without making any assumptions on retention times of DRAM cells, which may cause reliability issues (Section 3).

9. CONCLUSION

Our analysis of DDR4 DRAM's new Fine Granularity Refresh feature shows that there is no one-size-fits-all refresh option across the applications that we have used in our study. This makes our proposed *Adaptive Refresh (AR)* mechanism a simple yet effective way to leverage the best FGR mode in each application and phase within the application.

For high-density DRAM systems, we have identified a phenomenon that we call *command queue seizure*, whereby the memory controller's command queue seizes up because it is full with commands to a rank that is being refreshed. To attack this problem, we have proposed two complementary mechanisms called *Delayed Command Expansion (DCE)* and *Preemptive Command Drain (PCD)* which increase the number of issuable DRAM commands in the scheduler's command queue when a refresh operation is underway.

Once our proposed DCE and PCD mechanisms are in place, DDR4's FGR becomes redundant in most cases, except in highly memory-sensitive applications, where the use of AR does provide some additional benefit. In all, the proposed mechanisms yield significant performance gains with respect to traditional refresh at both normal and extended DRAM operating temperatures.

10. ACKNOWLEDGEMENTS

We thank Saugata Ghose for valuable feedback. This work was supported in part by NSF Award CNS-0720773.

11. REFERENCES

[1] *JEDEC DDR4 SDRAM Standard*, 2012. `http://www.jedec.org/standards-documents/docs/jesd79-4`.

[2] ASHRAE Technical Committee. 2011 Thermal Guidelines for Data Processing Environments - Expanded Data Center Classes and Usage Guidance. `http://www.eni.com/green-data-center/it_IT/static/pdf/ASHRAE_1.pdf`.

[3] V. Aslot and R. Eigenmann. Quantitative performance analysis of the SPEC OMPM2001 benchmarks. *Scientific Programming*, 11(2):105–124, 2003.

[4] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga. NAS parallel benchmarks. Technical Report RNR-94-007, NASA Ames Research Center, March 1994.

[5] M. Ghosh and H. S. Lee. Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3D die-stacked DRAMs. In *Proceedings of the 40th Intl. Symp. on Microarchitecture*, 2007.

[6] U. Kang, H. Chung, S. Heo, D. Park, H. Lee, J. H. Kim, S. Ahn, S. Cha, J. Ahn, D. Kwon, J. Lee, H. Joo, W. Kim, D. H. Jang, N. Kim, J.-H. Choi, T. Chung, J. Yoo, J. Choi, C. Kim, and Y. Jun. 8 Gb 3-D DDR3 DRAM using through-silicon-via technology for quasi-non-volatile DRAM. In *IEEE Journal of Solid State Circuits*, 2010.

[7] C. A. Kilmer, K. H. Kim, W. E. Maule, and V. Patel. Memory system with a programmable refresh cycle. United States Patent Application #0151131 A1, 2012.

[8] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. RAIDR: Retention-aware intelligent dram refresh. In *ISCA*, 2012.

[9] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. Flikker: Saving DRAM refresh-power through critical data partitioning. In *ASPLOS*, 2011.

[10] J. Pisharath, Y. Liu, W. Liao, A. Choudhary, G. Memik, and J. Parhi. NU-MineBench 2.0. Technical Report CUCIS-2005-08-01, Northwestern University, August 2005.

[11] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos. SESC simulator. `http://sesc.sourceforge.net`, January 2005.

[12] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. In *ISCA*, 2000.

[13] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. DRAMSim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters*, 10(1):16–19, Jan. 2011.

[14] B. Sinharoy, R. Kalla, W. J. Starke, H. Q. Le, R. Cargnoni, J. A. Van Norstrand, B. J. Ronchetti, J. Stuecheli, J. Leenstra, G. L. Guthrie, D. Q. Nguyen, B. Blaner, C. F. Marino, E. Retter, and P. Williams. IBM POWER7 multicore server processor. *IBM Journal of Research and Technology*, 55(3):1–29, 2011.

[15] K. Sohn, T. Na, I. Song, Y. Shim, W. Bae, S. Kang, D. Lee, H. Jung, S. Hyun, H. Jeoung, K. W. Lee, J. Park, J. Lee, B. Lee, I. Jun, J. Park, J. Park, H. Choi, S. Kim, H. Chung, Y. Choi, D. Jung, B. Kim, J. Choi, S. Jang, C. Kim, J. Lee, and J. Choi. A 1.2v 30nm 3.2Gb/s/pin 4Gb DDR4 SDRAM with dual-error detection and PVT-tolerant data-fetch scheme. In *ISSCC*, 2012.

[16] S. P. Song. Method and system for selective DRAM refresh to reduce power consumption. United States Patent #6094705, 2000.

[17] J. Stuecheli, D. Kaseridis, H. C. Hunter, and L. K. John. Elastic refresh: Techniques to mitigate refresh penalties in high density memory. In *MICRO*, 2010.

[18] R. K. Venkatesan, S. Herr, and E. Rotenberg. Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile DRAM. In *HPCA*, 2006.

[19] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *ISCA*, 1995.

[20] J. Worrel. Intel to introduce DDR4 memory with Haswell-EX server platform. In *http://fudzilla.com*, Apr. 2012.

[21] J. Yoon and G. Tressler. Advanced flash technology status, scaling trends and implications to enterprise SSD technology enablement. In *Flash Memory Summit*, 2012.